MARTE based modeling approach for Partial Dynamic Reconfigurable FPGAs

Imran Rafiq Quadri, Samy Meftali, Jean-Luc Dekeyser

MARTE based modeling approach for Partial Dynamic Reconfigurable FPGAs

Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser,
INRIA LILLE NORD EUROPE - LIFL - University of Lille - CNRS, Lille, France
{Imran.Quadri, Samy.Meftali, Jean-Luc.Dekeyser}@lifl.fr

***Abstract*— As System-on-Chip (SoC) architectures become pivotal for designing embedded systems, the SoC design complexity continues to increase exponentially necessitating the need to find new design methodologies. In this paper we present a novel SoC co-design methodology based on Model Driven Engineering using the MARTE (Modeling and Analysis of Real-time and Embedded Systems) standard. This methodology is utilized to model fine grain reconfigurable architectures such as FPGAs and extends the standard to integrate new features such as Partial Dynamic Reconfiguration supported by modern FPGAs. The goal is to carry out modeling at a high abstraction level expressed in UML (Unified Modeling Language) and following transformations of these models, automatically generate the code necessary for FPGA implementation.**

I. Introduction

Modern System-on-chips (SoCs) have become an integral part of designing embedded systems for targeting intensive parallel computation applications. Continuous advances in SoC technology permit to increase the number of hardware resources on a single chip while in parallel the targeted application domains: such as multimedia video codes, software defined radio, radar/sonar detection systems are becoming more sophisticated leading to a significant gap between design productivity and verification of these complex systems. An important challenge is to find efficient design methodologies that address the issues such as related to expressing parallelism.

Model Driven Engineering [1] (MDE) can be viewed as a *High Level Design Flow* and effectively resolves the various issues linked with SoC co-design. In a high level synthesis (HLS) flow, the underlying implementation details are usually hidden from the user resulting in advantages such as decrease in time to market and fabrication costs. However, usually the specifications are written in C/C++ or other text based languages and concepts such as related to parallelism and hierarchy are not clearly evident making the necessary modifications and extensions quite difficult. In contrast, in MDE, the system is modeled at a high level allowing several abstraction levels for different designers each focusing on a particular domain space. The UML (Unified Modeling Language) graphical language increases system comprehensibility and permits relations between concepts defined at different levels. Users provide high abstraction level descriptions of their systems and can identify the internal concepts such as task/data parallelism and hierarchy easily. The graphical nature of these specifications allows for their reuse, modification, maintenance and extension.

Modern FPGAs support the emerging feature of Partial Dynamic Reconfiguration [2] (PDR) allowing specific portions of FPGA to be reconfigured on the fly, hence time-sharing the available hardware resources. Moreover, PDR allows task swapping depending upon the application needs and Quality-Of-Service (QoS) requirements.

MARTE [3] (Modeling and Analysis of Real-Time and Embedded Systems) is an industry standard of the Object Management Group (OMG) for model-driven development of embedded systems and for SoC co-design. It provides the capability to model software, hardware and their relations, along with extensions for performance and scheduling analysis. This standard while rich in concepts unfortunately lacks certain aspects for FPGA modeling.

GASPARD [4] is a MARTE compliant SoC co-design environment dedicated specially towards parallel hardware and software co-design allowing to move from high level MARTE specifications to an executable platform. It exploits the parallelism included in repetitive constructions of hardware elements or regular constructions such as application loops.

The contribution of this paper is to present part of a complete design flow with an extended version of the MARTE standard for general modeling of FPGAs. Our concepts allow us to introduce PDR in MARTE for modeling all types of FPGAs supporting this feature. Finally, using the MDE model transformations, this methodology can be used to bridge the gap between high abstraction levels and implementation details to automatically generate the necessary code required for the generation of bitstream(s) for FPGA implementation.

The rest of this paper is organized as follows. Section 2 provides an overview of MDE and model transformations. Section 3 describes GASPARD and MARTE. Section 4 gives a summary of related works followed by PDR concepts. Section 6 defines our methodology for modeling PDR supported FPGAs. This paper finishes with a case study in section 7 followed by a conclusion.

II. Model Driven Engineering

MDE revolves around three focal concepts. *Models*, *Metamodels* and *Transformations*. A model is composed of concepts and relations and is an abstraction of reality. Concepts are “things” and relations are the “links” between these things in a real world scenario. A model can be observed from different point of views (views in MDE). A metamodel is a collective sum of concepts and relations for defining a

model and defines the model syntax as a language defines its grammar. Each model is then said to *conform* to its metamodel.

A model transformation is a compilation process that transforms a *source* model into a *target* model allowing to move from a higher abstract model to a lower detailed model. The source and target models conform to their respective metamodels. A model transformation is based on a set of *rules* that identify concepts in a source metamodel in order to create enriched concepts in the target metamodel. This separation allows to extend and maintain the compilation process. Each rule can be independently modified and new rules extend the compilation process. The transformations carry out refinements moving from high abstraction levels to low levels for code generation. At each intermediate level, implementation details are added to the compilation process. The advantage of this approach is that it allows to define several model transformations from the same abstraction level but targeted to different lower levels, offering opportunities to generate several implementations from a specification.

III. GASPARD Environment and MARTE

GASPARD [4] is a MDE based SoC co-design graphical environment and is a subset of the industry approved MARTE standard. MARTE allows a clear *separation of concerns* between the hardware and the software components which is of pivotal significance in SoC conception. Our environment uses the MARTE allocation mechanism (*Alloc* package) that permits to link the independent hardware and software models (for e.g. mapping of a task or data onto a processor or a memory respectively). GASPARD also relies heavily upon the *Repetitive Structure Modeling (RSM)* annex which is based on a MoC (Model of Computation) inspired from ArrayOL [5] that describes the potential parallelism in a system (parallel computations in the application software part and parallel structure of its hardware architecture in a compact manner). GASPARD currently targets *control and data flow oriented intensive signal processing (ISP) applications* (such as multimedia video codecs, radar/sonar detection applications, high performance applications). The applications targeted in GASPARD are widely encountered in SoC design and respect the semantics of ArrayOL [5].

GASPARD also benefits from the notion of a *Deployment* model level [6] that links every elementary component (the basic building block of all components) to an existing code facilitating Intellectual Property (IP) reuse. This level provides IP information for model transformations forming a compilation chain to transform the high abstraction level models (application, architecture and allocation) for different domains (currently GASPARD targets formal verification, simulation, high performance computing and synthesis). This last concept is currently not present in MARTE and is a potential extension to allow a flow from high level modeling to automatic code generation. We are currently in the process of integrating a control aspect [7] with this level which will allow to link an elementary component with several IPs (several possible implementations) and afterwards via model transformations convert this control aspect into the state machine code to be implemented in the reconfigurable controller in the FPGA automatizing part of the reconfiguration management.

IV. Related Works

ROSES [8] is a Multiprocessor SoC (MPSoC) specification and design environment but with a disadvantage as it does not conform to MDE concepts and as compared to GASPARD, starts from a low level description equivalent to our deployment level. In contrast, [9] uses the MDE approach for the design of a Software Defined Radio (SDR), but does not utilize the MARTE standard as proposed by OMG. While works such as [10] are focused on generating VHDL from UML state machines, they fail to integrate the MDE concepts for HW/SW co-design and are not capable to manage ISP applications. MILAN [11] is another MDE based project for SoC co-design but is not MARTE compliant. Only [12] comes close to our intended methodology by using the MDE concepts for creation of hardware accelerators. However the MARTE layer is absent and the reconfiguration is carried out in a static manner and there is no notion of PDR. While there are lots of related tools, works and projects; we have only detailed some and have not given an exhaustive summary. Also we have not included here the works specifically related to PDR which we have previously detailed in [13]. To the best of our knowledge, only our methodology takes into account the four domain spaces: SoC co-design, MDE, MARTE and PDR which is the novelty of our design flow.

V. Basic PDR related concepts

Currently only Xilinx FPGAs support PDR. Xilinx initially proposed two methodologies followed by the *Early Access Partial Reconfiguration (EAPR)* [14] flow. *Bus macros* are used to ensure the proper routing between the static and dynamic parts before and after reconfiguration. The *Internal Reconfiguration Access Port (ICAP)* [15] is present in nearly all Xilinx FPGAs and permits to read/write the FPGA configuration memory at run-time. In combination with the ICAP and the characteristics of *glitchless dynamic reconfiguration* supported by Xilinx FPGAs, a *Reconfiguration controller* (PowerPC or Microblaze) can be placed inside the FPGA to build a self controlling dynamically reconfigurable system.

VI. Modeling of Partial Dynamic Reconfigurable FPGAs

We first present the design flow to model and implement PDR supported FPGAs (Figure.1) which is an extension of the works presented in [12] (addition of MARTE layer, control at deployment level and the aspects of PDR). In this paper we only present the first model level of this flow (modeling of application, architecture and the allocation). Using model transformations, we will extend our work to link each modeled component with IP(s) at the Deployment model level (level 2). Afterwards, the RTL model level will provide detailed modeling information for the concepts modeled at level 1 and 2 such as the reconfiguration controller and reconfigurable hardware accelerator. Each of these model levels correspond to their respective metamodels. Finally, from the RTL level

we will be able to automatically generate the code for the FSM part of the Reconfiguration controller (to be implemented in a processor) and the reconfigurable portion (level 4). This will allow the creation of bitstreams for FPGA implementation using commercial tools and the usual synthesis design flows. An important restriction of our flow is that we have to adhere to the Xilinx technologies as currently only Xilinx FPGAs supports the PDR feature. While many extensions and PDR methodologies exist, we have chosen to respect the Xilinx EAPR flow as it is openly available and flexible enough to be modified for other PDR methodologies.

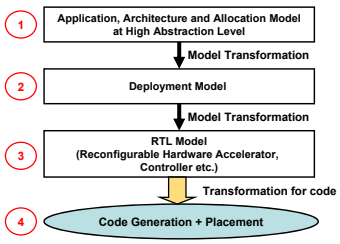

Fig. 1. *The complete design flow*

A. Basic MARTE Hardware concepts

In MARTE, The basic hardware concepts are presented in the *Hardware Resource Model (HRM)*. HRM can be viewed in different manners, either a functional view (*HwLogical* sub package), a physical view (*HwPhysical* sub package) or a merge of the two. These two subpackages derive from a root package called *HwGeneral* that revolves around the concept of a *HwResource* which defines a generic MARTE hardware entity that can be composed of other HwResource(s). This concept is then further enriched according to the functional or physical specifications. The functional view defines hardware resources as either *computing*, *storage*, *communication*, *timing* or *device* resources. The physical view represents hardware resources as physical components with details about their shape, size and power consumption among other attributes. Currently GASPARD supports only the logical view but we have integrated both the physical and merged views in the framework for modeling PDR featured architectures. The HRM also exploits the Non-Functional Properties (NFP) package of MARTE. This package introduces an accurate value specification language for supporting complex expressions for specifying non-functional properties as well as quantitative annotations with measurement units.

B. MARTE modifications for PDR concepts

We first examined the HRM package of MARTE and found it to be lacking in certain aspects. The *HwComputing* subpackage in the HRM functional view defines a set of active processing resources central for an execution platform. A *HwComputingResource* symbolizes an active processing resource that can be specialized as either a processor (*HwProcessor*), an ASIC (*HwASIC*) or a PLD (*HwPLD*). An FPGA is represented by the HwPLD stereotype, it can contain a RAM memory (*HwRAM*) (as well as other HwResources) and is characterized by a technology (SRAM, Antifuse etc.). The cell organization of the FPGA is characterized by the number of rows and columns, but also by the type of architecture (Symmetrical array, row based etc.). These concepts are sufficient enough for an abstract FPGA description. However the concepts related to representing a processor are not sufficient for a complex SoC design in which a processor can either be implemented as a softcore IP or integrated as a hardcore IP. We thus add the attribute **imtype** (Implementation_Type) that is flexible enough to define a processor implementation as either **Hardcore** or **Softcore** and adaptable with future evolution using the **Other** and **Undefined** types. Figure.2 shows only the simplified modeling description of the modified HwComputing subpackage related to a processor implementation.

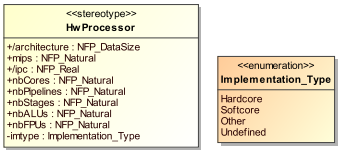

Fig. 2. *Modified version of the HwProcessor concept*

The second modification relates to the physical (HwLayout) package that revolves around the *HwComponent* concept which is an abstraction of any real hardware entity based on its physical attributes. HwComponent can be specialized as either *HwChip* (e.g. a processor), *HwChannel* (e.g a bus), *HwPort* (e.g. an interface), *HwCard* (e.g. a motherboard) or a *HwUnit* (a hardware resource that does not fall into the preceding four categories). In order to specify the nature of the area for a PDR featured architecture (either static or dynamically reconfigurable), we have introduced the attribute **areatype** (Areatype) which can be either **Static**, **DynamicReconf** or typed as **Other** to adapt to future evolution. Although this concept can be implemented as a functional property, we have chosen to implement it in the physical view. Figure.3 shows the simplified overview of our modified HwComponent concept.

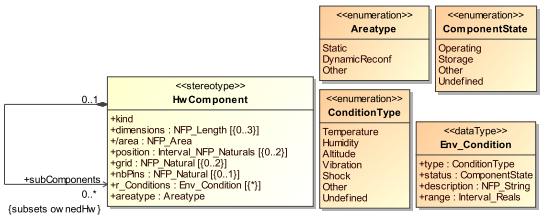

Fig. 3. *Modified version of the HwComponent concept*

These are the 2 general concepts that we have introduced at the specification level of the MARTE standard and could generally benefit other frameworks and methodologies and can be extended. We now present the specific concepts related to FPGA and PDR in our methodology.

We then present a example of a classical PDR supported Xilinx FPGA. We have taken the Virtex-II Pro on a XUP Board (as shown in figure.4) as a reference as it is a popular choice for implementing PDR. The architecture consists of a Reconfiguration Controller (a PowerPC in this case) connected

to a 64-bit PLB bus and communicates with the slower slave peripherals (connected to the 32-bit OPB bus) via a PLB to OPB Bridge. The peripherals connected to the OPB bus are detailed as follows. A SystemACE controller for accessing the partial bitstreams placed in an external onboard Compact Flash (CF) card. A SDRAM controller for a DDR SDRAM present onboard that permits the partial bitstreams to be preloaded from the CF during initialization. An ICAP is present in the form of an OPB peripheral (OPBHwICAP) permitting partial reconfiguration using the read-modify-write mechanism [15]. This static portion is connected to a Reconfigurable Hardware Accelerator (RHA) via bus macros. Also it was an implementation choice to connect the RHA with the OPB bus. The concepts such as PowerPC, PLB and OPB buses, PLB to OPB Bridge, CF and SDRAM memories can be easily explained using the current MARTE HRM concepts. However the peripherals, bus macros, ICAP and RHA require an extended and more detailed conception.

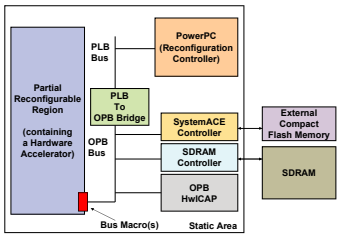

Fig. 4. *Block Diagram of the architecture of our reconfigurable system*

The *HwCommunication* subpackage in the HRM functional view defines the basic concepts for hardware communication. *HwMedia* is the central concept defining a communication resource capable of data transfer with a theoretical bandwidth. It can be controlled by a *HwArbiter* and connected to other HwMedia(s) by means of a *HwBridge*. A *HwEndpoint* defines a connection point of a HwResource and can be defined as an interface (e.g. pin or port). *HwBus* illustrates a specific wired channel with particular functional attributes. These concepts are sufficient and abstract enough to define all kind of communication resources. Some of the other common HRM concepts that we utilize are *HwComputingResource* (to describe a general computing resource) from the *HwComputing* package, *HwRAM* and *HwROM* from the *HwMemory* package (for RAM and ROM concepts), *HwStorageManager* from the *HwStorageManager* package (for a memory controller), *HwClock* from the *HwTiming* package (to specify a clock and *HwIO* from the *HwIO* package (for an I/O resource).

Xilinx provides the notion of an Intellectual Property Interface (IPIF) which is a hardware bus wrapper specially designed to ease IP core interfacing with the IBM Coreconnect bus. As all peripherals in our architecture consist of the IPIF wrapper and an IP core, this is a vital modeling concept. The IPIF has two basic attributes: a **mode** which can be either **Master**, **Slave** or **Master/Slave**, and **type** that determines the protocol of IPIF adapted for a particular bus. It can be either **PLB**, **OPB** or extensible using **Other** or **Undefined** types. We avoided adding detailed information related to the protocols offered by IPIF to simplify its definition at the high abstraction level. The IPIF itself is typed *HwEndpoint* to denote that it is a hardware wrapper providing an interface to the IP core. We thus add the notion of a hardware wrapper using MARTE. Figure.5 shows the modeling of the IPIF.

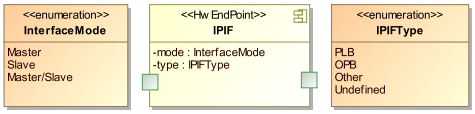

Fig. 5. *Modeling of the IPIF hardware wrapper*

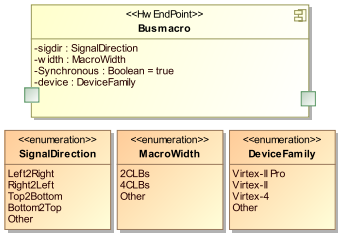

Fig. 6. *Modeling of a Bus macro*

The second modeling concept is that of Bus Macros (BMs). Although the EAPR flow now allows signals in the base design to pass through the reconfigurable regions without the use of bus macros, they are still essential in order to ensure the correct routing between the static and dynamic regions. They are CLB based in nature and provide a unidirectional 8-bit data transfer. Figure.6 shows the modeling of a Bus Macro (*Busmacro*) having four attributes. The **sigdir** attribute determines the direction of communication which can be **Left2Right** or **Right2Left** (for Virtex-II and Virtex-II Pro devices), as well as **Top2Bottom**, **Bottom2Top** or **Other** for Virtex-IV and other future PDR supported devices. The **width** attribute determines the CLB width of the bus macro (**2CLBs** or **4CLBs** width making it either a narrow or wide bus macro or use of **Other** for a user specified width). The **Synchronous** attribute determines if the bus macro is synchronous or not. We have assigned a default value of **true** to this attribute (as recommended by Xilinx). The final attribute **device** determines the targeted FPGA device family (either **Virtex-II Pro**, **Virtex-II**, **Virtex-4** or a newer device such as Virtex-5 using the **Other** type). The Bus Macro is typed *HwEndpoint* in order to illustrate that it is an interface.

We then carry out modeling of the OPB_HWICAP peripheral. It consists of an IPIF (**ic2opb**) connected to the HWICAP core (**hwicap**) (typed as *HwComputingResource*) and is itself defined as a *HwComputingResource*. The HWICAP core is itself composed of three components: an ICAP controller (**icapctrl**) and ICAP Primitive (**icap**) both typed as *HwComputingResource*(s) and a BlockRAM (**bram**) defined as *HwRAM* for stocking a configuration frame of FPGA memory. The BlockRAM contains a port having a multiplicity of 2 indicating that it is repeated two times. The *Reshape* connectors (as defined in the MARTE RSM package) are used in order to link the sub components of the HWICAP. The Reshape allows to represent complex link topologies in a simplified manner. Here, the Reshape connectors permits to specify accurately

which port (either the port of the ICAPController or the single port of the HWICAP itself) is connected to which repetition of the BlockRAM port as shown in figure.7. Also, the sub components of HWICAP have specific attributes (e.g. BlockRAM has 16Kbits memory) related to implementation details. We refer the reader to [15] for a detailed description of the HWICAP core.

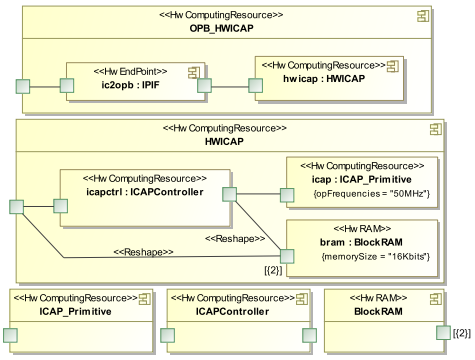

Fig. 7. *Modeling of the OPB HWICAP Peripheral*

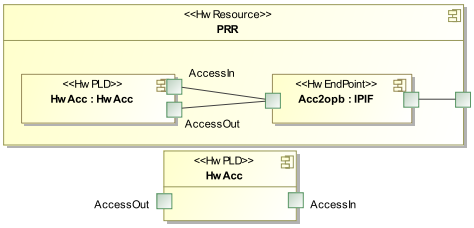

Fig. 8. *A Reconfigurable Hardware Accelerator*

Figure.8 illustrates the modeling of the Reconfigurable Hardware Accelerator (RHA). The PRR (Partial reconfigurable region) consists of a RHA (**HwAcc**) typed as *HwPLD* having ports **AccessOut** and **AccessIn** and an IPIF (**Acc2opb**). The PRR itself is of the *HwResource* type. The RHA is typed as *HwPLD* as it is reconfigurable, as compared to a typical hardware accelerator which can be seen as a *HwASIC* depending upon the designer's point of view.

Figure.9 finally illustrates our reconfigurable architecture (a XC2VP30 Virtex-II Pro chip) utilizing our proposed concepts in a merged functional/physical view. Each of the hardware components has two type definitions (the first representing the functional and the second representing the physical one). The XC2VP30 chip consists of a PowerPC PPC405 (**ppc_0**) connected via a PLB bus (**plb**) to the slave peripherals: the OPB_HWICAP (**opbhwicap**), the OPB_SysAceCtrl (**opbsys_ac_ctr**), the OPB_SDRAMCtrl (**opbsdram_ctr**) and the PRR (**prr**) via the OPB bus (**opb**). The PLB2OPB_Bridge (**plb2opb**) connects the two buses, while Busmacro(s) (**bm1** and **bm0** having types Left2Right and Right2Left respectively) connect the OPB bus to the PRR. Each of the busmacros are instantiated twice as indicated by the multiplicity of 2 on both **bm0** and **bm1** respectively. Also the OPB bus has a **slave_a** port with a multiplicity of 3 which allows the bus to connect to the peripherals (opbsys_ac_ctr, opbsdram_ctr and opbhwicap), we have used Reshape connectors to determine which peripheral is connected to which repetition of the slave port. Similarly we have used Reshape connectors to determine the accurate connections between the bus macros and the ports of OPB and PRR. Although we could have used a single slave port on OPB with an appropriate multiplicity to include the topology of bus macros, this is avoided in order to reduce the design complexity. Finally, the XC2VP30 contains two HwEndPoint(s) interfaces, **toCompactFlash** and **toSDRAM** to connect **opbsys_ac_ctr** and **opbsdram_ctr** to the compact flash and the SDRAM memories respectively. The Attributes introduced by us and those by default in MARTE allow the designer to specify general attributes of each component at the highest abstraction level (e.g. ppc_0 having a frequency of 300 MHz). While the MARTE modifications and the PDR modeling may seem trivial, however this is done explicitly in order to elevate the abstraction levels and decrease the implementation details at these higher levels.

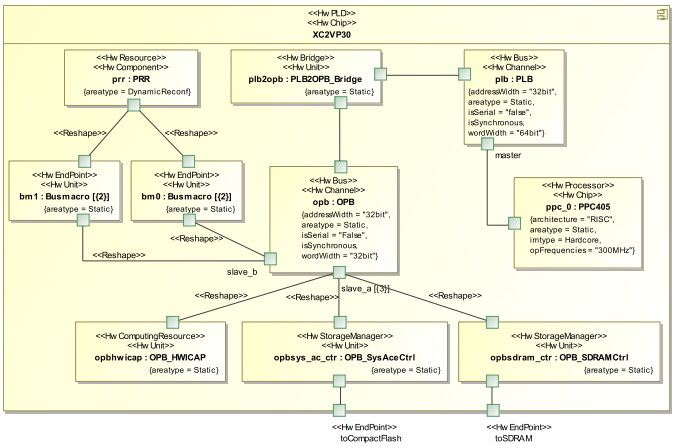

Fig. 9. *Modeling of our PDR Architecture*

VII. Case Study

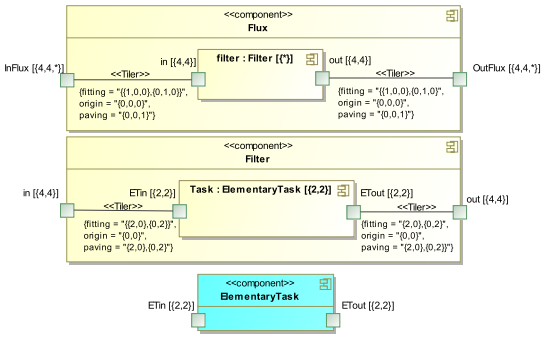

Fig. 10. *Model of an Image Filter task*

A case study of a complete SoC model is presented here to illustrate our methodology. We present here only a simple image filter application, however other multimedia applications such as H.263 encoder [6] can be implemented using GASPARD and our design flow. The modeled application *MainApplication* is an academic grayscale 4x4 pixel image filter application (producing 8-bit images). It consists of three

tasks (application components) : An image sensor PictureGen (**pg**), the image filter task Flux (**tasks**) (shown in figure.10) and an output PictureRead (**pr**). The Flux component is comprised of a Filter component (**filter**) (repeating infinitely as shown by the multiplicity of *). The Filter component itself contains an elementary application component ElementaryTask (**Task**) being repeated four times (having a multiplicity of 2 by 2). The Tiler connectors are used to describe the tiling of produced and consumed arrays by a pattern mechanism [5]. The Flux component can be implemented in different manners depending upon IP characteristics and the controller modeled at deployment can create the corresponding FSM code.

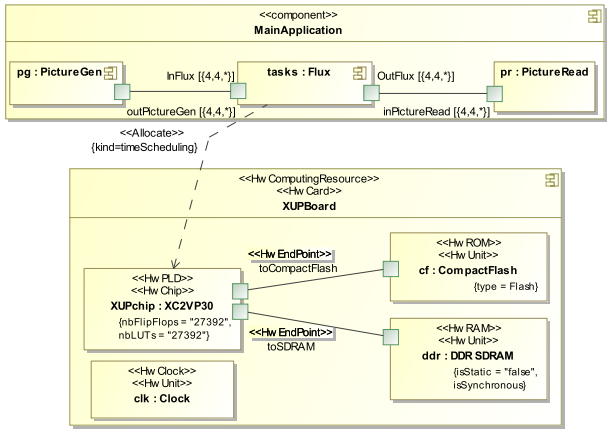

Fig. 11. *Allocation Level 1*

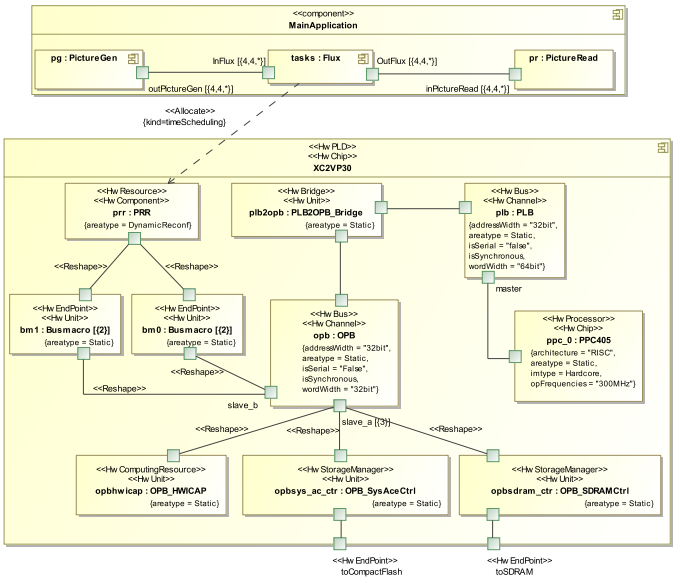

Fig. 12. *Allocation Level 2*

Figures.11 and 12 represent the allocation of the application on to the architecture. In Figure.11 the model of the whole application is shown allocated to the XC2VP30 chip (**XUPchip**) on an XUPBoard via the *Allocate* type allocation. Currently GASPARD only supports spacial distribution (static scheduling at compilation time due to the nature of targeted applications), however due to the nature of PDR and related applications; we integrate the *timeScheduling* (dynamic scheduling) nature of allocation as defined in MARTE. Figure.12 presents a detailed view of the allocation illustrating the mapping of the application onto the PRR portion. Due to space limitations we have not presented the last level of allocation in which the image filter task is finally placed on a hardware accelerator **HwAcc**. The XUPBoard also contains a Clock (**clk**) and the CompactFlash (**cf**) and DDR SDRAM memories (**ddr**). The concepts introduced in our approach can be modified and extended to manipulate other types of PDR supported architectures such as introduced in [16],[17] and [18] validating our modeling approach.

VIII. Conclusions

The paper describes a new design methodology to model PDR featured FPGAs using the MDE approach and the MARTE standard. The existing MARTE standard has been modified to add concepts for general FPGA modeling and specifically for PDR integration. This methodology can also be adapted to serve other Xilinx based fine grain reconfigurable architectures. In future works, we will detail the model transformations and the low level models for automatic generation of the FSM code of the reconfiguration controller and the reconfigurable hardware accelerator for FPGA implementation.

References

[1] Planet MDE, *Portal of the Model Driven Engineering Community*, 2007, http://www.planetmde.org.

[2] P. Lysaght et al, "Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," in *FPL'06*, 2006.

[3] Object Management Group, *OMG MARTE Standard*, 2007, http://www.omgmarte.org.

[4] The DaRT team, "GASPARD Design Environment," 2008, https://gforge.inria.fr/projects/gaspard2.

[5] P. Boulet, "Array-OL Revisited, Multidimensional Intensive Signal Processing Specification," 2007.

[6] R.B. Atitallah et al, "Multilevel MPSoC simulation using an MDE approach," in *SoCC 2007*, 2007.

[7] H. Yu et al, "Safe Design of High-Performance Embedded Systems in an MDE framework," *NASA/Springer ISSE Journal*, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11334-008-0059-y

[8] W. Cesario et al, "Component-Based Design Approach for Multicore SoCs," *DAC'02*, vol. 00, p. 789, 2002.

[9] G. Gailliard et al, "Transaction level modelling of SCA compliant software defined radio waveforms and platforms PIM/PSM," in *DATE'07*, 2007.

[10] R. Damasevicius et al, "Application of UML for hardware design based on design process model," in *ASP-DAC'04*, 2004, pp. 244–249.

[11] S. Mohanty et al, "Rapid DSE of heterogeneous embedded systems using symbolic search and multi-granular simulation," in *LCTES/Scopes 2002*, 2002.

[12] S. Le Beux et al, "A Model Driven Engineering Design Flow to generate VHDL," in *International ModEasy'07 Workshop*, 2007.

[13] I.R. Quadri and S. Meftali and J-L. Dekeyser, "An MDE Approach for Implementing Partial Dynamic Reconfiguration in FPGAs," in *IP'07*, 2007.

[14] Xilinx, "Early Access Partial Reconfigurable Flow," 2006, http://www.xilinx.com/support/prealounge/protected/index.htm.

[15] B. Blodget et al, "A lightweight approach for embedded reconfiguration of FPGAs," in *DATE'03*, 2003.

[16] A. Tumeo et al, "A Self-Reconfigurable Implementation of the JPEG Encoder," *ASAP 2007*, pp. 24–29, 2007.

[17] K. Paulsson et al, "Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGAs," *FPL 2007*, pp. 351–356, 2007.

[18] C. Claus et al, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," *IPDPS 2007*, pp. 1–7, 2007.